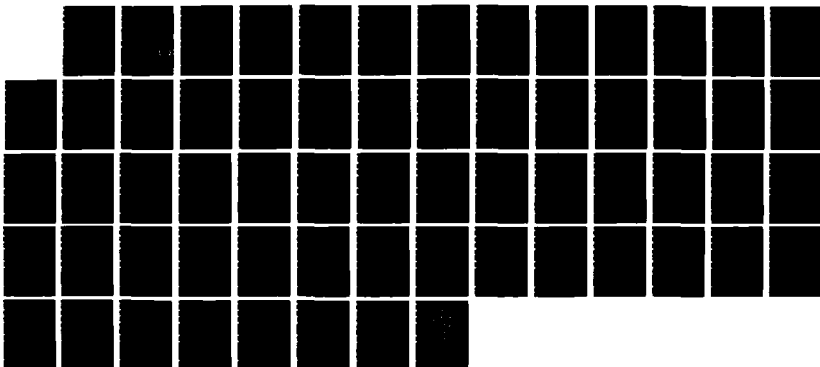ADA (TRADEMARK) COMPILER VALIDATION SUMMARY REPORT(U)
NATIONAL COMPUTING CENTRE LTD MANCHESTER (ENGLAND)
30 APR 87

F/G 12/5   NL
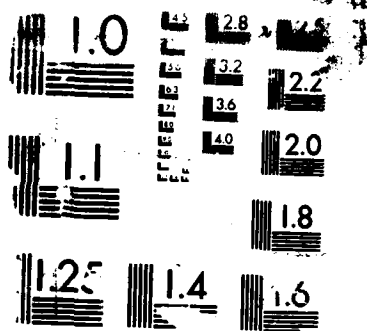
MICROCOPY RESOLUTION TEST CHART

AD-A189 235

AVF Control Number: AVF-VSR-90502/11

Ada* COMPILER
VALIDATION SUMMARY REPORT
ALSYS
AlsyCOMP_014, V2.0
Host: SUN-3/160
Target: IBM 370/3084Q

Completion of On-Site Testing
30 April 1987

Prepared By
The National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
UK

DTIC
ELECTE
DEC 2 8 1987
S   D
H

Prepared For
Ada Joint Program Office
United States Department of Defense
Washington, D.C.
USA

*Ada is a registered trademark of the United States Government
(Ada Joint Program Office).

87 12 14 162

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETEING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Ada Compiler Validation Summary Report: ALSYS. AlsyCOMP_014, V2.0 Host: SUN-3/160 Target: IBM 370/3084Q		5. TYPE OF REPORT & PERIOD COVERED 30 Apr.'87 to 30 Apr.'88
		6 PERFORMING ORG. REPORT NUMBER -
7. AUTHOR(s) The National Computing Centre Ltd. -		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS The National Computing Centre Ltd. Oxford Rd., Manchester, M1 7ED, UK		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL		12. REPORT DATE 30 Apr. '87
		13. NUMBER OF PAGES 58
14. MONITORING AGENCY NAME & ADDRESS*(If different from Controlling Office)* The National Computing Centre Ltd.		15. SECURITY CLASS *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20. If different from Report)*

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS *(Continue on reverse side if necessary and identify by block number)*

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See Attached.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

```
++++++++++++++++++++++++++++++
+                            +
+    Place NTIS form here    +
+                            +
++++++++++++++++++++++++++++++
```

Ada* Compiler Validation Summary Report:

Compiler Name: AlsyCOMP_014, V2.0
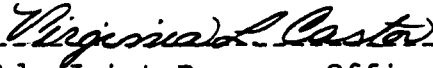
Host: Target:
    SUN-3/160   IBM 370 3084Q
    under   under
    BSD UNIX   MVS
    4.2   3.2

Testing Completed 30 April 1987 using ACVC 1.8

This report has been reviewed and is approved.

The National Computing Centre Ltd
Vony Gwillim
Oxford Road
Manchester
M1 7ED

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

----
*Ada is a registered trademark of the United States Government
(Ada Joint Program Office).

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the AlsyCOMP_014, V2.0 using Version 1.8 of the Ada* Compiler Validation Capability (ACVC). The AlsyCOMP_014, V2.0 is hosted on a SUN-3/160 operating under BSD UNIX 4.2. Programs processed by this compiler may be executed on a IBM 370 3084Q operating under MVS 3.2.

On-site testing was performed 27 April 1987 through 30 April 1987 at ALSYS Ltd, Henley-on-Thames, under the direction of the National Computing Centre (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2246 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as 134 executable tests that make use of floating-point precision exceeding that supported by the implementation were not processed. After the 2246 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 36 of the processed tests determined to be inapplicable; The remaining 2210 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	105	269	352	243	161	97	134	262	121	32	217	217	2210
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	11	56	68	4	0	0	5	0	9	0	1	16	170
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution. Keywords: Ada programming languages.

1-1

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

. To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

. To attempt to identify any unsupported language constructs required by the Ada Standard.

. To determine that the implementation-dependent behaviour is allowed by the Ada Standard

Testing of this compiler was conducted by NCC under the direction of the AVF according to policies and procedures established by the Ada Validation Organisation (AVO). On-site testing was conducted from 27 April 1987 through 30 April 1987 at ALSYS Ltd, Henley-on- Thames.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. 552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organisations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

> Ada Information Clearinghouse
> Ada Joint Program Office
> OUSDRE
> The Pentagon, Rm 3D-139 (Fern Street)
> Washington DC 20301-3081

or from:

> Ada Validation Facility
> The National Computing Centre Ltd
> Oxford Road
> Manchester
> M1 7ED
> United Kingdom

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

> Ada Validation Organization
> Institute for Defense Analyses
> 1801 North Beauregard
> Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementer's Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC   The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.

Ada Standard   ANSI/MIL-STD-1815A, February 1983.

Applicant   The agency requesting validation.

AVF   The National Computing Centre Ltd. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.

AVO   The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.

Compiler   A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test   A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host   The computer on which the compiler resides.

Inapplicable   A test that uses features of the language that a compiler
test   is not required to support or may legitimately support in
               a way other than the one expected by the test.

Passed test   A test for which a compiler generates the expected
               result.

Target   The computer for which a compiler generates code.

Test   A program that checks a compiler's conformity regarding a
               particular feature or features to the Ada Standard.   In
               the context of this report, the term is used to designate
               a single test, which may comprise one or more files.

Withdrawn   A test found to be incorrect and not used to check
               conformity to test the Ada language specification.   A
               test may be incorrect because it has an invalid test
               objective, fails to meet its test objective, or contains
               illegal or erroneous use of the language.

1.5   ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC.   The ACVC
contains both legal and illegal Ada programs structured into six test
classes: A, B, C, D, E, and L.   The first letter of a test name
identifies the class to which it belongs.   Class A, C, D, and E tests
are executable, and special program units are used to report their
results during execution.   Class B tests are expected to produce
compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully
compiled and executed.   However, no checks are performed during
execution to see if the test objective has been met.   For example, a
Class A test checks that reserved words of another language (other
than those already reserved in the Ada language) are not treated as
reserved words by an Ada compiler.   A Class A test is passed if no
errors are detected at compile time and the program executes to
produce a PASSED message.

Class B tests check that a compiler detects illegal language usage.
Class B tests are not executable.   Each test in this class is compiled
and the resulting compilation listing is examined to verify that every
syntax or semantic error in the test is detected.   A Class B test is
passed if every illegal construct that it contains is detected by the
compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capabilities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimization allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended
to ensure that the tests are reasonably portable without modification.
For example, the tests make use of only the basic set of 55
characters, contain lines with a maximum length of 72 characters, use
small numeric values, and place features that may not be supported by
all implementations in separate tests. However, some tests contain
values that require the test to be customized according to
implementation-specific values--for example, an illegal file name. A
list of the values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and
demonstrate conformity to the Ada Standard either meeting the pass
criteria given for the test or by showing that the test is
inapplicable to the implementation. Any test that was determined to
contain an illegal language construct or an erroneous language
construct is withdrawn from the ACVC and, therefore, is not used in
testing a compiler. The tests withdrawn at the time of validation are
given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: AlsyCOMP_014, V2.0

ACVC Version: 1.8

Certification Expiration Date: DD Month 1988

Host Computer:

Machine	:	SUN-3/160
Operating System:		BSD UNIX 4.2
Memory Size:		6 M

Target Computer:

Machine	:	IBM 370 3084Q
Operating System:		MVS 3.2
Memory Size:		2 M region

Communications Network: Magnetic tape was used for the transfer of executable programs from host to target.

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

. Capacities.

The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

. Predefined types.

This implementation supports the additional predefined types SHORT_INTEGER, SHORT_FLOAT, and LONG_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

. Based literals.

An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

. Array Types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/ or SYSTEM.MAX_INT.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

2-2

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

2-3

. Functions

An implementation may allow the declaration of a parameterless
function and an enumeration literal having the same profile in
the same immediate scope, or it may reject the function
declaration. If it accepts the function declarations, the use
of the enumeration literal's identifier denotes the function.
This implementation rejects the declarations. (See test
E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support
representation clauses. If a representation clause is not
supported, then the implementation must reject it. While the
operation of representation clauses is not checked by Version
1.8 of the ACVC, they are used in testing other language
features. This implementation rejects 'SIZE and 'STORAGE_SIZE
for tasks, 'STORAGE_SIZE for collections, and 'SMALL clauses.
Enumeration representation clauses appear not to be supported.
(See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is not supported for procedures. The pragma
INLINE is not supported for functions. (See tests CA3004E and
CA3004F.)

. Input/Output.

The package SEQUENTIAL_IO can be instantiated with
unconstrained array types and record types with discriminants.
The package DIRECT_IO can be instantiated with unconstrained
array types and record types with discriminants without
defaults, however, USE_ERROR will be raised if IO operations
are attempted. (See tests AE2101C, AE2101H, CE2201D, CE2201E,
and CE2401D.)

An existing text file can be opened in OUT_FILE mode, can be
created in OUT_FILE mode, and can be created in IN_FILE mode.
(See test EE3102C.)

Only one internal file can be associated with each external
file for text I/O for both reading and writing. (See tests
CE3111A.E (5 tests).)

Only one internal file can be associated with each external
file for sequential I/O for both reading and writing. (See
tests CE2107A..F (6 tests).)

Only one internal file can be associated with each

external file for direct I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

Temporary sequential files are given a name. Temporary direct files are given a name. Temporary files given names are not deleted when they are closed. (See tests CE2108A and CE2108C.)

. Generics.

Generic subprogram bodies can only be compiled in separate compilations provided that no instantiations of the corresponding generic occur prior to the compilation of the generic body. (See test CA2009F.)

Generic package bodies can only be compiled in separate compilations provided that no instantiations of the corresponding generic occur prior to the compilation of the generic body. (See tests CA2009C and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests.   When validation testing
of AlsyCOMP_014,V2.0 was performed, 19 tests had been withdrawn.   The
remaining 2380 tests were potentially applicable to this validation.
The   AVF determined that 170 tests were inapplicable to   this
implementation,   and that the 2210 applicable tests were passed by the
implementation.

The AVF concludes that the testing results demonstrate acceptable
conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	862	1208	17	11	44	2210
Failed	0	0	0	0	0	0	0
Inapplicable	1	5	160	0	2	2	170
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	105	269	352	243	161	97	134	262	121	32	217	217	2210
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	11	56	68	4	0	0	5	0	9	0	1	16	170
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B332C3C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 170 tests were inapplicable for the reasons indicated:

. C34001E, B52004D, B55B09C, and C55B07A use LONG_INTEGER which is not supported by this compiler.

. C55B16A makes use of an enumeration representation clause containing noncontiguous values which is not supported by this compiler.

3-2

. B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

. BA2001E requires that duplicate names of subunits with a common ancestor be detected at compilation-time. This compiler correctly detects the error at link-time, and the AVO rules that such behaviour is acceptable.

. C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

. C87B62A..C (3 tests) use length clauses which are not supported by this compiler. The length clause is rejected during compilation.

. CA2009C, CA2009F, and BC3205D compile generic subunits in separate compilation files. Separate compilation of generic specifications and bodies is not supported by this compiler when instantiations precede the generic bodies.

. CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

. CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

. AE2101H, CE2401D use instantiation of package DIRECT_IO with unconstrained array types which is not supported by this compiler.

. CE2107A..F (6 tests), CE2110B, CE3111A..E (5 tests), CE3114B and CE3115A attempt to associate more than one external file with the same internal file, which is not supported by this implementation.

. The following 134 tests make use of floating-point precision that exceeds the maximum of 18 supported by the implementation:

                C24113O..Y (11 tests)
                C35705O..Y (11 tests)
                C35706O..Y (11 tests)
                C35707O..Y (11 tests)
                C35708O..Y (11 tests)
                C35802O..Y (11 tests)
                C45241O..Y (11 tests)
                C45321O..Y (11 tests)
                C45421O..Y (11 tests)
                C45424O..Y (11 tests)
                C45521O..Z (12 tests)
                C45621O..Z (12 tests)

3.6   SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B
test because of compiler error recovery, then the test is split into a
set of smaller tests that contain the undetected errors.   These splits
are then compiled and examined.   The splitting process continues until
all errors are detected by the compiler or until there is exactly one
error per split.   Any Class A, Class C, or Class E test that cannot be
compiled and executed because of its size is split into a set of
smaller subsets that can be processed.

Splits were required for 15 Class B tests.

B26005A		
B32202A	B45102A	B95069A
B32202B	B61012A	B95069B
B32202C	B62001B	
B33001A	B62001C	
B37004A	B62001D	
B43201D	B91004A	

3.7   ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8
produced by AlsyCOMP_014,V2.0 was submitted to the AVF by the
applicant for review.   Analysis of these results demonstrated that the
compiler successfully passed all applicable tests, and the compiler
exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of AlsyCOMP_014,V2.0 using ACVC Version 1.8 was conducted on-
site by a validation team from the AVF.   The configuration consisted
of a SUN-3/160 operating under BSD UNIX 4.2, and a IBM 370 3084Q
target operating under MVS,V3.2.   The host and target computers were
linked via magnetic tape.

A magnetic tape containing all tests was taken on-site by the
validation team for processing.   The magnetic tape contained tests
that make use of implementation-specific values were customized
before being written to the magnetic tape.   Tests requiring splits
during the prevalidation testing were not included in their split form
on the magnetic tape.   The contents of the magnetic tape were loaded
first onto a VAX 750 computer, where the required splits were
performed using the VAX editor (EDT) using prepared command scripts.
The processed source files were then transferred to the host computer
via an Ethernet connection.

After the test files were loaded to disk, the full set of tests was compiled and linked on the SUN-3/160, and all executable tests were run on the IBM 370 3084Q. Object files were bound on the host computer and bound object modules were transferred to the target computer via magnetic tape. Bound object modules were linked on the target computer and run. Results were transferred to the host computer via a data-link using SNA protocol and printed from a VAX 750 connected to the host computer via Ethernet.

The compiler was tested using command scripts provided by ALSYS Ltd and reviewed by the validation team.

Tests were compiled, linked and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at ALSYS Ltd, Henley-on-Thames on 27 April 1987 and departed after testing was completed on 30 April 1987.

APPENDIX A

COMPLIANCE STATEMENT

ALSYS Ltd has submitted the following
compliance statement concerning the
AlsyCOMP_014,V2.0.

Compliance Statement

Base Configuration:

>   Compiler: AlsyCOMP_014, V2.0

>   Test Suite: Ada* Compiler Validation Capability, Version 1.8

>   Host Computer:

>>   Machine: SUN-3/160

>>   Operating System: BSD UNIX
>>   4.2

>   Target Computer:

>>   Machine: IBM 370 3084Q
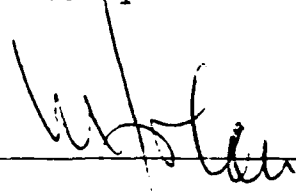
>>   Operating System: MVS
>>   3.2

>   Communications Network: Magnetic tape was used for the
>   transfer of executable programs
>   from host to target

ALSYS Ltd. has made no deliberate extensions to the Ada language
standard.

ALSYS Ltd. agrees to the public disclosure of this report.

ALSYS Ltd. agrees to comply with the Ada trademark policy, as
defined by the Ada Joint Program Office.

_____   Date: ___30/4/87___

ALSYS Ltd.
M L J Jordan
Marketing Director

_____
*Ada is registered trademark of the United States Government
(Ada Joint Program Office).

A-2

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to
implementation-dependent pragmas, to certain machine-dependent
conventions as mentioned in chapter 13 of MIL-STD-1815A, and to
certain allowed restrictions on representation classes. The
implementation-dependent characteristics of the AlsyCOMP_014, V2.0 are
described in the following sections which discuss topics one through
eight as stated in Appendix F of the Ada Language Reference Manual
(ANSI/MIL-STD-1815A). The implementation-specific portions of the
package STANDARD are also included in this appendix.

Package STANDARD is

   .

   ...

   type INTEGER is -2_147_483_648   .. 2_147_483_647 ;
   type SHORT_INTEGER is range -32_768 .. 32_767   ;

   type FLOAT is digits 15 range -(1.0-2.0**-56)*2.0**252 ..
                                  (1.0-2.0**-56)*2.0**252;
   type SHORT_FLOAT is digits 6 range
      -(1.0-2.0**-24)*2.0**252   .. (1.0-2.0**-24)*2.0**252   ;
   type LONG_FLOAT is digits 18 range
      -(1.0-2.0**-112)*2.0**252   .. (1.0-2.0**-112)*2.0**252   ;
   type DURATION is delta 1.0E-4 range -86_400.0..86_400.0;

   end STANDARD;

Alsys IBM 370 Ada[*] Compiler

Appendix F

Implementation – Dependent Characteristics

Version 2.0

Alsys S.A.
29, Avenue de Versailles
78170 La Celle St. Cloud. France

Alsys, Inc.
1432 Main Street
Waltham, MA 02154. U.S.A.

Alsys Ltd.
Partridge House. Newtown Road
Henley-on-Thames,
Oxfordshire RG9 1EN. U.K.

[*] Ada is a registered trademark of the U.S. Government, Ada Joint Program Office

PREFACE

This *Alsys IBM 370 Ada Compiler Appendix F* is for programmers, software engineers, project managers, educators and students who want to develop an Ada program for any IBM System 370 (30XX or 43XX) processor that runs MVS.

This appendix is a required part of the *Reference Manual for the Ada Programming Language*, ANSI MIL-STD 1815A, February 1983 (throughout this appendix, citations in square brackets refer to this manual). It assumes that the user is already familiar with the MVS operating system, and has access to the following IBM documents:

OS VS2 MVS Overview, GC28-0984

OS/VS2 System Programming Library: Job Management, GC28-1303

OS/VS2 MVS JCL, GC28-1350

IBM System/370: Principles of Operation, GA22-7000

IBM System/370 System Summary, GA22-7001

TABLE OF CONTENTS

Appendix F

Implementation-Dependent Characteristics

This appendix summarises the implementation-dependent characteristics of the Alsys IBM 370 Ada Compiler.

The sections of this appendix are as follows:

1. The form, allowed places, and effect of every implementation-dependent pragma.

2. The name and type of every implementation-dependent attribute.

3. The specification of the package SYSTEM.

4. The list of all restrictions on representation clauses.

5. The conventions used for any implementation-generated names denoting implementation-dependent components.

6. The interpretation of expressions that appear in address clauses, including those for interrupts.

7. Any restrictions on unchecked conversions.

8. Any implementation-dependent characteristics of the input-output packages.

9. Characteristics of numeric types.

10. Other implementation-dependent characteristics.

11. Compiler limitations.

The name *Ada Run-Time Executive* refers to the run-time library routines provided for all Ada programs. These routines implement the Ada heap, exceptions, tasking control, and other utility functions.

1 Implementation-Dependent Pragmas

Ada programs can interface with subprograms written in assembler or other languages through the use of the predefined pragma INTERFACE [13.9] and the implementation-defined pragma INTERFACE_NAME.

1.1 INTERFACE

Pragma INTERFACE specifies the name of an interfaced subprogram and the name of the programming language for which calling and parameter passing conventions

will be generated. Pragma INTERFACE takes the form specified in the *Reference Manual*:

   pragma INTERFACE *(language_name, subprogram_name)*;

where

- *language_name* is the name of the other language whose calling and parameter passing conventions are to be used.

- *subprogram_name* is the name used within the Ada program to refer to the interfaced subprogram.

The only language name currently accepted by pragma INTERFACE is ASSEMBLER.

The language name used in the pragma INTERFACE does not necessarily correspond to the language used to write the interfaced subprogram. It is used only to tell the Compiler how to generate subprogram calls, that is, which calling conventions and parameter passing techniques to use. ASSEMBLER is used to refer to the standard IBM 370 calling and parameter passing conventions. The programmer can use the language name ASSEMBLER to interface Ada subprograms with subroutines written in any language that follows the standard IBM 370 calling conventions.

Calling Conventions

The contents of all of the general purpose registers must be left unchanged by the call, except register 0, which is used for returning results. On entry to the subprogram, register 13 contains the address of a register save area provided by the caller.

Registers 15 and 14 contain the entry point address and return address, respectively, of the called subprogram.

An interfaced subprogram should have the following structure:

```
    STM   R14.R12,12(R13)           -- save registers

        .
        .                            -- subprogram body
        .

    LM    R14,R12,12(R13)           -- restore registers
    PR    R14                        -- return to caller
```

Any registers which are altered by the execution of the subprogram should be saved as the first action upon entry to the subprogram, using a single ST or STM instruction. This enables the Ada Run-Time Executive to treat any interruption occuring during the execution of the body of the subprogram as the implementation-defined Ada exception SPURIOUS_ERROR being raised at the point of call of the subprogram. This exception is not visible outside the Ada Run-Time Executive, and hence cannot be handled by the Ada program.

Parameter-Passing Conventions

On entry to the subprogram, register 1 contains the address of a parameter address list. Each word in this list is an address corresponding to a parameter. The last word in the list has its bit 0 (sign bit) set.

For actual parameters which are literal values, the address is that of a copy of the value of the parameter; for all other parameters it is the address of the parameter object. Interfaced subprograms have no notion of parameter modes; hence parameters whose addresses are passed are not protected from modification by the subprogram, even though they may be formally declared to be of mode in.

No consistency checking is performed between the subprogram parameters declared in Ada and the corresponding parameters of the interfaced subprogram. It is the programmer's responsibility to ensure correct access to the parameters.

Parameter Representations

This section describes the representation of values of the types that can be passed as parameters to an interfaced subprogram.

Integer Types [3.5.4]

Ada integer types occupy 16 (SHORT_INTEGER) or 32 (INTEGER) bits. An INTEGER subtype falling within the range of SHORT_INTEGER is implemented as a SHORT_INTEGER in 16 bits.

Enumeration Types [3.5.1]

Values of an Ada enumeration type are represented internally as unsigned values representing their position in the list of enumeration literals defining the type. The first literal in the list corresponds to a value of zero.

Enumeration types with 256 elements or fewer are represented in 8 bits, those with more than 256 elements in 16 bits. The maximum number of values an enumeration type can include is 65536 (2**16).

The ADA predefined type CHARACTER [3.5.2] is represented in 8 bits, using the standard ASCII codes [C].

Floating Point Types [3.5.7, 3.5.8]

Ada floating-point values occupy 32 (SHORT_FLOAT), 64 (FLOAT) or 128 (LONG_FLOAT) bits, and are held in IBM 370 (short, long or extended floating point) format.

Fixed Point Types [3.5.9, 3.5.10]

Ada fixed-point types are managed by the Compiler as the product of a signed *mantissa* and a constant *small*. The mantissa is implemented as a 16 or 32 bit integer value. *Small* is a compile-time quantity which is the power of two equal or immediately inferior to the delta specified in the declaration of the type.

The attribute MANTISSA is defined as the smallest number such that:

$$2 ** \text{MANTISSA} >= \max (\text{abs} (\text{upper\_bound}), \text{abs} (\text{lower\_bound})) / \text{small}$$

The size of a fixed point type is:

MANTISSA	Size
1 .. 15	16 bits
16 .. 31	32 bits

Fixed point types requiring a MANTISSA greater than 31 are not supported.

Access Types [3.8]

Values of access types are represented internally by the 31-bit address of the designated object. Note that bit 0 (the sign bit) of the 32-bit word holding a non-null access value may be set or clear, depending upon certain conventions used by the Ada Run-Time Executive, and must be preserved. The value zero is used to represent **null**.

Array Types [3.6]

Ada arrays are passed by reference; the value passed is the address of the first element of the array. When an array is passed as a parameter to an interfaced subprogram, the usual consistency checking between the array bounds declared in the calling program and the subprogram is not enforced. It is the programmer's responsibility to ensure that the subprogram does not violate the bounds of the array.

Values of the predefined type STRING [3.6.3] are arrays, and are passed in the same way: the address of the first character in the string is passed. Elements of a string are represented in 8 bits, using the standard ASCII codes.

Record Types [3.7]

Ada records are passed by reference, by passing the address of the first component of the record. However, unlike arrays, the individual components of a record may be reordered internally by the Ada compiler. Moreover, if a record contains discriminants or components having a dynamic size, implicit components may be added to the record. Thus the exact internal structure of the record in memory may not be inferred directly from its Ada declaration.

Restrictions on Interfaced subprograms

The Ada Run-Time Executive uses the SPIE (SVC 14) macro. Interfaced subprograms should avoid use of this facility, or else restore interruption processing to its original state before returning to the Ada program. Failure to do so may lead to unpredictable results.

Similarly, interfaced subprograms must not change the program mask in the Program Status Word (PSW) of the machine without restoring it before returning.

1.2 INTERFACE_NAME

Pragma INTERFACE_NAME associates the name of an interfaced subprogram, as declared in Ada, with its name in the language of origin. If pragma INTERFACE_NAME is not used, then the two names are assumed to be identical. This pragma takes the form

pragma INTERFACE_NAME (*subprogram_name*, *string_literal*);

where

- *subprogram_name* is the name used within the Ada program to refer to the interfaced subprogram.

- *string_literal* is the name by which the interfaced subprogram is referred to at link-time.

The use of INTERFACE_NAME is optional, and is not needed if a subprogram has the same name in Ada as in the language of origin. It is useful, for example, if the name of the subprogram in its original language contains characters that are not permitted in Ada identifiers. Ada identifiers can contain only letters, digits and underscores, whereas the IBM 370 linkage editor/loader allows external names to contain other characters, e.g. the plus or minus sign. These characters can be specified in the *string_literal* argument of the pragma INTERFACE_NAME.

The pragma INTERFACE_NAME is allowed at the same places of an Ada program as the pragma INTERFACE [13.9]. However, the pragma INTERFACE_NAME must always occur after the pragma INTERFACE declaration for the interfaced subprogram.

In order to conform to the naming conventions of the IBM 370 linkage editor/loader, the link-time name of an interfaced subprogram will be truncated to 8 characters and converted to upper case.

The Ada Run-Time Executive contains several external identifiers. All such identifiers begin with the string "ALSYS". Accordingly, names prefixed by "ALSYS", in any combination of upper and lower case, should be avoided by the user.

Example

```
package SAMPLE_DATA is
     function SAMPLE_DEVICE (X : INTEGER) return INTEGER;
     function PROCESS_SAMPLE (X : INTEGER) return INTEGER;
private
     pragma INTERFACE (ASSEMBLER, SAMPLE_DEVICE);
     pragma INTERFACE (ASSEMBLER, PROCESS_SAMPLE);
     pragma INTERFACE_NAME (PROCESS_SAMPLE, "PSAMPLE");
end SAMPLE_DATA.
```

1.3   Other Pragmas

No other implementation-dependent pragmas are supported in the current version of this compiler.

2   Implementation-Dependent Attributes

There are no implementation-dependent attributes.

3   Specification of the Package SYSTEM


```
package SYSTEM is

      type NAME is (IBM_370);

      SYSTEM_NAME : constant NAME := NAME'FIRST;
      MIN_INT         : constant := -(2**31);
      MAX_INT        : constant :=  2**31-1;
      MEMORY_SIZE : constant := 2**24;

      type ADDRESS is range MIN_INT .. MAX_INT;

      STORAGE_UNIT : constant := 8;
      MAX_DIGITS    : constant := 18;
      MAX_MANTISSA : constant := 31;
      FINE_DELTA    : constant := 2#1.0#e-31;
      TICK              : constant := 0.01;
      NULL_ADDRESS : constant ADDRESS := 0;

      subtype PRIORITY is INTEGER range 1 .. 10;

      -- These subprograms are provided to perform
      -- READ/WRITE operations in memory.

      generic
              type ELEMENT_TYPE is private;
      function FETCH (FROM : ADDRESS) return ELEMENT_TYPE;
```

```
generic
      type ELEMENT_TYPE is private
   procedure STORE (INTO   ADDRESS, OBJECT   ELEMENT_TYPE);

end SYSTEM;
```

The generic function FETCH may be used to read data objects from given addresses in store. The generic procedure STORE may be used to write data objects to given addresses in store.

4   Restrictions on Representation Clauses

Representation clauses [13.1] are not supported by this version of the Alsys IBM 370 Ada Compiler. Any program containing a representation clause is rejected at compilation time. The pragma PACK [13.1] is also not supported. However, its presence in a program does not in itself make the program illegal; the Compiler will simply issue a warning message and ignore the pragma.

5   Conventions for Implementation-Generated Names

There are no implementation-generated names [13.4] in the current version of the Alsys IBM 370 Ada Compiler.

The following predefined library units cannot be recompiled:

```
ALSYS_ADA_RUNTIME
ALSYS_BASIC_IO
ALSYS_BINARY_IO
ALSYS_COMMON_IO
ALSYS_FILE_MANAGEMENT
ALSYS_SYS_IO
CALENDAR
DIRECT_IO
EBCDIC
IO_EXCEPTIONS
OS_ENV
SEQUENTIAL_IO
STANDARD
SYSTEM
TEXT_IO
UNCHECKED_CONVERSION
UNCHECKED_DEALLOCATION
```


6   Address Clauses

Address clauses [13.5] are not supported in this version of the Alsys IBM 370 Ada Compiler.

7 Restrictions on Unchecked Conversions

Unchecked conversions [13.10.2] are allowed only between types which have the same value for their 'SIZE attribute.

8 Input-Output Packages

The predefined input-output packages SEQUENTIAL_IO [14.2.3], DIRECT_IO [14.2.5], and TEXT_IO [14.3.10] are implemented as described in the Language Reference Manual, as is the package IO_EXCEPTIONS [14.5], which specifies the exceptions that can be raised by the predefined input-output packages.

The package LOW_LEVEL_IO [14.6], which is concerned with low-level machine-dependent input-output, has not been implemented.

8.1 Specifying External Files

The NAME parameter supplied to the Ada procedure CREATE [14.2.1] must represent an MVS dataset name (DSNAME). The NAME parameter supplied to the OPEN procedure [14.2.1] may represent a DSNAME or a DDNAME.

Files

An MVS dataset name as specified in the Ada NAME parameter may be given in any of the following forms:

```
OPEN (F, NAME => "UNQUALIFIED.NAME", ...);

OPEN (F, NAME => "'FULLY.QUALIFIED.NAME'", ...);

OPEN (F, NAME => "UNQUALIFIED.PDS (MEMBER)", ...);

OPEN (F, NAME => "'FULLY.QUALIFIED.PDS (MEMBER)'", ...);
```

An unqualified name (not enclosed in apostrophes) is first prefixed by the name (if any) given as the QUALIFIER parameter in the program PARM string when the program is run.

The QUALIFIER parameter may be specified as in the following example:

```
  STEP20 EXEC PGM=IEB73 PARM='QUALIFIER(PAYROLL.ADA)'
```

A fully qualified name (enclosed in single quotes) is not so prefixed. The result of the NAME function is always in the form of a fully qualified name, i.e. enclosed in apostrophes.

Members of partitioned datasets are specified within parentheses.

The file name parameter may also be a DDNAME (see below) in the case of OPEN.

FORM Parameter

The FORM parameter comprises a (possibly empty) set of attributes (the FORM parameter may, of course, be given as a null string [14.2.1]) formulated according to the lexical rules of [2], separated by commas. Attributes are comma-separated; blanks may be inserted between lexical elements as desired. In the descriptions below the meanings of *natural, positive*, etc., are as in Ada; attribute keywords (represented in upper case) are identifiers [2.3] and as such may be specified without regard to case.

USE_ERROR is raised if the FORM parameter is illegal.

The attributes are as follows:

File sharing attribute

This attribute allows control over the sharing of one external file between several internal files within a single program. In effect it establishes rules for subsequent OPEN and CREATE calls which specify the same external file. If such rules are violated or if a different file sharing attribute is specified in a later OPEN or CREATE call, USE_ERROR will be raised. The syntax is as follows:

NOT_SHARED | SHARED => *access_mode*

where

access_mode ::= READERS | SINGLE_WRITER | ANY

A file sharing attribute of:

NOT_SHARED

implies only one internal file may access the external file.

SHARED => READERS

imposes no restrictions on internal files of mode IN_FILE, but prevents any internal files of mode OUT_FILE or INOUT_FILE being associated with the external file.

SHARED => SINGLE_WRITER

is as SHARED => READERS, but in addition allows a single internal file of mode OUT_FILE or INOUT_FILE.

SHARED => ANY

places no restrictions on external file sharing.

The default is SHARED => NOT_SHARED.

Record size attribute

By default, records are output according to the following rules:

- for TEXT_IO and SEQUENTIAL_IO, variable-length record files (RECFM = V).

- for DIRECT_IO, fixed-length record files (RECFM = F).

In the case of DIRECT_IO for constrained types the record size is determined by the size of the type with which the package is instantiated. The user can specify the record size attribute to force the representation of the Ada element in output records of a given byte size. Of course, such a specified record size must not be smaller than ELEMENT_TYPE'SIZE / SYSTEM.STORAGE_UNIT; DATA_ERROR will be raised if this rule is violated.

There is just one case in which the record size attribute is mandatory, that of DIRECT_IO for mode OUT_FILE or INOUT_FILE for unconstrained types. In the absence of the record size attribute in this case, USE_ERROR will be raised (although the package may be instantiated without error).

If the record size attribute is specified, fixed-length records (RECFM = F) will be generated.

In the case of TEXT_IO, output lines will be padded to the requisite length with spaces; this fact should be borne in mind when re-reading files generated using TEXT_IO with the record size attribute set.

In the case of DIRECT_IO of unconstrained types, the length of each item precedes the binary image of the item itself in the external file; it is held in 2 or 4 bytes depending on the maximum size of the item.

The syntax of the record size attribute is as follows:

    RECORD_SIZE => *natural*

where *natural* is a size in bytes.

The default is

    RECORD_SIZE => ELEMENT_TYPE'SIZE / SYSTEM.STORAGE_UNIT

for DIRECT_IO of constrained types.

    RECORD_SIZE => 0

(meaning variable-length records) otherwise.

Carriage control

This attribute applies to TEXT_IO only, and is intended for files destined to be sent to a printer.

For a file of mode OUT_FILE, this attribute causes the output procedures of TEXT_IO to place a carriage control character as the first character of every output record: '1' (skip to channel 1) if the record follows a page terminator, or blank (skip to next line) otherwise. Subsequent characters are output as normal as the result of calls of the output subprograms of TEXT_IO.

For a file of mode IN_FILE, this attribute causes the input procedures of TEXT_IO to interpret the first character of each record as a carriage control character, as described in the previous paragraph. Carriage control characters are not explicitly returned as a result of an input subprogram, but will (for example) affect the result of END_OF_PAGE.

The user should naturally be careful to ensure the carriage control attribute of a file of mode IN_FILE has the same value as that specified when creating the file.

The syntax of the carriage control attribute is as follows:

    CARRIAGE_CONTROL => *boolean*

The default is CARRIAGE_CONTROL => FALSE.

DDNAME attribute

This attribute affects the semantics of the NAME parameter.

If the DDNAME attribute is specified, the NAME parameter is taken to be the name of a DD statement which the user must have provided in the JCL to run the Ada program.

CREATE will raise USE_ERROR if the DDNAME attribute is specified in its FORM parameter.

If DELETE is called for a file opened with the DDNAME attribute of the FORM parameter having been specified, USE_ERROR will be raised, but the file will be closed.

The syntax of the DDNAME attribute is as follows:

    DDNAME => *boolean*

The default is DDNAME => FALSE.

8.2   Text Terminators

Line terminators [14.3] are not implemented using a character, but are implied by the end of physical record.

Page terminators [14.3] are implemented using the EBCDIC character 0C (hexadecimal).

File terminators [14.3] are not implemented using a character, but are implied by the end of physical file.

The user should avoid the explicit output of the character ASCII.FF [C]. If the user explicitly outputs the character ASCII.LF, this is treated as a call of NEW_LINE [14.3.4].

8.3 EBCDIC and ASCII

All I/O using TEXT_IO is performed using ASCII/EBCDIC translation. CHARACTER and STRING values are held internally in ASCII but represented in external files in EBCDIC. For SEQUENTIAL_IO and DIRECT_IO no translation takes place, and the external file contains a binary image of the internal representation of the Ada element.

It should be noted that the EBCDIC character set is larger than the (7 bit) ASCII and that the use of EBCDIC and ASCII control characters may not produce the desired results when using TEXT_IO (the input and output of control characters is in any case not defined by the Ada language [14.3]). Furthermore, the user is advised to exercise caution in the use of BAR (|) and SHARP (#), which are part of the lexis of Ada; if their use is prevented by translation between ASCII and EBCDIC, EXCLAM (!) and COLON (:), respectively, should be used instead [2.10].

Various translation tables exist to translate between ASCII and EBCDIC. The predefined package EBCDIC is provided to allow access to the translation facilities used by TEXT_IO and OS_ENV.

The specification of this package is as follows:

```
package EBCDIC is

    type EBCDIC_CHARACTER is (
        nul,                            -- 0 = 0h
        soh,                            -- 1 = 1h
        stx,                            -- 2 = 2h
        etx,                            -- 3 = 3h
        E_4,
        ht,                             -- 5 = 5h
        E_6,
        del,                            -- 7 = 7h
        E_8,
        E_9,
        E_A,
        vt,                             -- 11 = 0Bh
        np,                             -- 12 = 0Ch
        cr,                             -- 13 = 0Dh
        so,                             -- 14 = 0Eh
        si,                             -- 15 = 0Fh
        dle,                            -- 16 = 10h
        dc1,                            -- 17 = 11h
        dc2,                            -- 18 = 12h
```

```
dc3,                                -- 19 = 13h
E_14
nl,                                 -- 21 = 15h
bs,                                 -- 22 = 16h
E_17,
can,                                -- 24 = 18h
em,                                 -- 25 = 19h
E_1A,
E_1B,
E_1C,
gs,                                 -- 29 = 1Dh
rs,                                 -- 30 = 1Eh
us,                                 -- 31 = 1Fh
E_20,
E_21,
fs,                                 -- 34 = 22h
E_23,
E_24,
E_25,
etb,                                -- 38 = 26h
esc,                                -- 39 = 27h
E_28,
E_29,
E_2A,
E_2B,
E_2C,
enq,                                -- 45 = 2Dh
ack,                                -- 46 = 2Eh
bel,                                -- 47 = 2Fh
E_30,
E_31,
syn,                                -- 50 = 32h
E_33,
E_34,
E_35,
E_36,
eot,                                -- 55 = 37h
E_38,
E_39,
E_3A,
E_3B,
dc4,                                -- 60 = 3Ch
nak,                                -- 61 = 3Dh
E_3E,
sub,                                -- 63 = 3Fh
sp,                                 -- 64 = 40h
E_41,
E_42,
E_43,
E_44,
E_45,
E_46,
```

```
        E_47,
        E_48,
        E_49,
        E_4A,
        '.',                          -- 75 = 4Bh
        '<',                          -- 76 = 4Ch
        '(',                          -- 77 = 4Dh
        '+',                          -- 78 = 4E?
        '|',                          -- 79 = 4Fh
        '&',                          -- 80 = 50H
        E_51,
        E_52,
        E_53,
        E_54,
        E_55,
        E_56,
        E_57,
        E_58,
        E_59,
        '!',                          -- 90 = 5Ah
        '$',                          -- 91 = 5Bh
        '*',                          -- 92 = 5Ch
        ')',                          -- 93 = 5Dh
        ';',                          -- 94 = 5Eh
        '^',                          -- 95 = 5Fh
        '-',                          -- 96 = 60h
        '/',                          -- 97 = 61h
        E_62,
        E_63,
        E_64,
        E_65,
        E_66,
        E_67.
        E_68,
        E_69,
        E_6A,
        ',',                          --107 = 6Bh
        '%',                          --108 = 6Ch
        '_',                          --109 = 6Dh
        '>',                          --110 = 6Eh
        '?',                          --111 = 6Fh
        E_70,
        E_71,
        E_72,
        E_73,
        E_74,
        E_75,
        E_76,
        E_77,
        E_78,
        '`',                          --121 = 79h
        ':',                          --122 = 7Ah
```

```
    '#',                              --123 = 7Bh
    '@',                              --124 = 7Ch
    '''',                             --125 = 7Dh
    '=',                              --126 = 7Eh
    '"',                              --127 = 7Fh
    E_80,
    'a',                              --129 = 81h
    'b',                              --130 = 82h
    'c',                              --131 = 83h
    'd',                              --132 = 84h
    'e',                              --133 = 85h
    'f',                              --134 = 86h
    'g',                              --135 = 87h
    'h',                              --136 = 88h
    'i',                              --137 = 89h
    E_8A,
    E_8B,
    E_8C,
    E_8D,
    E_8E,
    E_8F,
    E_90,
    'j',                              --145 = 91h
    'k',                              --146 = 92h
    'l',                              --147 = 93h
    'm',                              --148 = 94h
    'n',                              --149 = 95h
    'o',                              --150 = 96h
    'p',                              --151 = 97h
    'q',                              --152 = 98h
    'r',                              --153 = 99h
    E_9A,
    E_9B,
    E_9C,
    E_9D,
    E_9E,
    E_9F,
    E_A0,
    '-',                              --161 = 0A1h
    's',                              --162 = 0A2h
    't',                              --163 = 0A3h
    'u'.                              --164 = 0A4h
    'v',                              --165 = 0A5h
    'w',                              --166 = 0A6h
    'x',                              --167 = 0A7h
    'y',                              --168 = 0A8h
    'z',                              --169 = 0A9h
    E_AA,
    E_AB,
    E_AC,
    '[',                              --173 = 0ADh
    E_AE,
```

```
E_AF,
E_B0
E_B1,
E_B2,
E_B3,
E_B4,
E_B5,
E_B6,
E_B7,
E_B8,
E_B9,
E_BA,
E_BB,
E_BC,
'|',                                          --189 = 0BDh
E_BE,
E_BF,
'{',                                          --192 = 0C0h
'A',                                          --193 = 0C1h
'B',                                          --194 = 0C2h
'C',                                          --195 = 0C3h
'D',                                          --196 = 0C4h
'E',                                          --197 = 0C5h
'F',                                          --198 = 0C6h
'G',                                          --199 = 0C7h
'H',                                          --200 = 0C8h
'I',                                          --201 = 0C9h
E_CA,
E_CB,
E_CC,
E_CD,
E_CE,
E_CF.
'}',                                          --208 = 0D0h
'J',                                          --209 = 0D1h
'K',                                          --210 = 0D2h
'L',                                          --211 = 0D3h
'M',                                          --212 = 0D4h
'N',                                          --213 = 0D5h
'O',                                          --214 = 0D6h
'P',                                          --215 = 0D7h
'Q',                                          --216 = 0D8h
'R',                                          --217 = 0D9h
E_DA,
E_DB,
E_DC,
E_DD,
E_DE,
E_DF,
'\',                                          --224 = 0E0h
E_E1,
'S',                                          --226 = 0E2h
```

```
         'T',                                        --227 = 0E3h
         'U',                                        - 228 = 0E4h
         'V',                                        --229 = 0E5h
         'W',                                        --230 = 0E6h
         'X',                                        --231 = 0E7h
         'Y',                                        --232 = 0E8h
         'Z',                                        --233 = 0E9h
         E_EA,
         E_EB,
         E_EC,
         E_ED,
         E_EE,
         E_EF,
         '0',                                        --240 = 0F0h
         '1',                                        --241 = 0F1h
         '2',                                        --242 = 0F2h
         '3',                                        --243 = 0F3h
         '4',                                        --244 = 0F4h
         '5',                                        --245 = 0F5h
         '6',                                        --246 = 0F6h
         '7',                                        --247 = 0F7h
         '8',                                        --248 = 0F8h
         '9',                                        --249 = 0F9h
         E_FA,
         E_FB,
         E_FC,
         E_FD,
         E_FE,
         E_FF);


SEL      : constant EBCDIC_CHARACTER := E_4;
RNL      : constant EBCDIC_CHARACTER := E_6;
GE       : constant EBCDIC_CHARACTER := E_8;
SPS      : constant EBCDIC_CHARACTER := E_9;
RPT      : constant EBCDIC_CHARACTER := E_A;
RES      : constant EBCDIC_CHARACTER := E_4;
ENP      : constant EBCDIC_CHARACTER := E_4;
POC      : constant EBCDIC_CHARACTER := E_17;
UBS      : constant EBCDIC_CHARACTER := E_1A;
CU1      : constant EBCDIC_CHARACTER := E_1B;
IFS      : constant EBCDIC_CHARACTER := E_1C;
DS       : constant EBCDIC_CHARACTER := E_20;
SOS      : constant EBCDIC_CHARACTER := E_21;
WUS      : constant EBCDIC_CHARACTER := E_23;
BYP      : constant EBCDIC_CHARACTER := E_24;
INP      : constant EBCDIC_CHARACTER := E_24;
LF       : constant EBCDIC_CHARACTER := E_25;
SA       : constant EBCDIC_CHARACTER := E_28;
SFE      : constant EBCDIC_CHARACTER := E_29;
SM       : constant EBCDIC_CHARACTER := E_2A;
SW       : constant EBCDIC_CHARACTER := E_2A;
CSP      : constant EBCDIC_CHARACTER := E_2B;
```

```
MFA          constant EBCDIC_CHARACTER := E_2C;
IR           constant EBCDIC_CHARACTER := E_33;
PP           constant EBCDIC_CHARACTER := E_34;
TRN          constant EBCDIC_CHARACTER := E_35;
NBS          constant EBCDIC_CHARACTER := E_36;
SBS          constant EBCDIC_CHARACTER := E_38;
IT           constant EBCDIC_CHARACTER := E_39;
RFF          constant EBCDIC_CHARACTER := E_3A;
CU3          constant EBCDIC_CHARACTER := E_3B;
RSP          constant EBCDIC_CHARACTER := E_41;
CENT         constant EBCDIC_CHARACTER := E_4A;
SHY          constant EBCDIC_CHARACTER := E_CA;
HOOK         : constant EBCDIC_CHARACTER := E_CC;
FORK         : constant EBCDIC_CHARACTER := E_CE;
NSP          : constant EBCDIC_CHARACTER := E_E1;
CHAIR        : constant EBCDIC_CHARACTER := E_EC;
EO           : constant EBCDIC_CHARACTER := E_FF;


E_0          : constant EBCDIC_CHARACTER := nul;
E_1          : constant EBCDIC_CHARACTER := soh;
E_2          : constant EBCDIC_CHARACTER := stx;
E_3          : constant EBCDIC_CHARACTER := etx;
E_5          : constant EBCDIC_CHARACTER := ht;
E_7          : constant EBCDIC_CHARACTER := del;
E_B          : constant EBCDIC_CHARACTER := vt;
E_C          : constant EBCDIC_CHARACTER := np;
E_D          : constant EBCDIC_CHARACTER := cr;
E_E          : constant EBCDIC_CHARACTER := so;
E_F          : constant EBCDIC_CHARACTER := si;
E_10         : constant EBCDIC_CHARACTER := dle;
E_11         : constant EBCDIC_CHARACTER := dc1;
E_12         : constant EBCDIC_CHARACTER := dc2;
E_13         : constant EBCDIC_CHARACTER := dc3;
E_15         : constant EBCDIC_CHARACTER := nl;
E_16         : constant EBCDIC_CHARACTER := bs;
E_18         : constant EBCDIC_CHARACTER := can;
E_19         : constant EBCDIC_CHARACTER := em;
E_1D         : constant EBCDIC_CHARACTER := gs;
E_1E         : constant EBCDIC_CHARACTER := rs;
E_1F         : constant EBCDIC_CHARACTER := us;
E_22         : constant EBCDIC_CHARACTER := fs;
E_26         : constant EBCDIC_CHARACTER := etb;
E_27         : constant EBCDIC_CHARACTER := esc;
E_2D         : constant EBCDIC_CHARACTER := enq;
E_2E         : constant EBCDIC_CHARACTER := ack;
E_2F         constant EBCDIC_CHARACTER := bel;
E_32         constant EBCDIC_CHARACTER := syn;
E_37         constant EBCDIC_CHARACTER := eot;
E_3C         constant EBCDIC_CHARACTER := dc4;
E_3D         constant EBCDIC_CHARACTER := nak;
E_3F         constant EBCDIC_CHARACTER := sub;
```

```
E_40        : constant EBCDIC_CHARACTER := sp;
E_4B        : constant EBCDIC_CHARACTER := '.';
E_4C        : constant EBCDIC_CHARACTER := '<';
E_4D        : constant EBCDIC_CHARACTER := '(';
E_4E        : constant EBCDIC_CHARACTER := '+';
E_4F        : constant EBCDIC_CHARACTER := '|';
E_50        : constant EBCDIC_CHARACTER := '&';
E_5A        : constant EBCDIC_CHARACTER := '!';
E_5B        : constant EBCDIC_CHARACTER := '$';
E_5C        : constant EBCDIC_CHARACTER := '*';
E_5D        : constant EBCDIC_CHARACTER := ')';
E_5E        : constant EBCDIC_CHARACTER := ';';
E_5F        : constant EBCDIC_CHARACTER := '^';
E_60        : constant EBCDIC_CHARACTER := '-';
E_61        : constant EBCDIC_CHARACTER := '/';
E_6B        : constant EBCDIC_CHARACTER := ',';
E_6C        : constant EBCDIC_CHARACTER := '%';
E_6D        : constant EBCDIC_CHARACTER := '_';
E_6E        : constant EBCDIC_CHARACTER := '>';
E_6F        : constant EBCDIC_CHARACTER := '?';
E_79        : constant EBCDIC_CHARACTER := '`';
E_7A        : constant EBCDIC_CHARACTER := ':';
E_7B        : constant EBCDIC_CHARACTER := '#';
E_7C        : constant EBCDIC_CHARACTER := '@';
E_7D        : constant EBCDIC_CHARACTER := ''';
E_7E        : constant EBCDIC_CHARACTER := '=';
E_7F        : constant EBCDIC_CHARACTER := '"';
E_81        : constant EBCDIC_CHARACTER := 'a';
E_82        : constant EBCDIC_CHARACTER := 'b';
E_83        : constant EBCDIC_CHARACTER := 'c';
E_84        : constant EBCDIC_CHARACTER := 'd';
E_85        : constant EBCDIC_CHARACTER := 'e';
E_86        : constant EBCDIC_CHARACTER := 'f';
E_87        : constant EBCDIC_CHARACTER := 'g';
E_88        : constant EBCDIC_CHARACTER := 'h';
E_89        : constant EBCDIC_CHARACTER := 'i';
E_91        : constant EBCDIC_CHARACTER := 'j';
E_92        : constant EBCDIC_CHARACTER := 'k';
E_93        : constant EBCDIC_CHARACTER := 'l';
E_94        : constant EBCDIC_CHARACTER := 'm';
E_95        : constant EBCDIC_CHARACTER := 'n';
E_96        : constant EBCDIC_CHARACTER := 'o';
E_97        : constant EBCDIC_CHARACTER := 'p';
E_98        : constant EBCDIC_CHARACTER := 'q';
E_99        : constant EBCDIC_CHARACTER := 'r';
E_A1        : constant EBCDIC_CHARACTER := '~';
E_A2        : constant EBCDIC_CHARACTER := 's';
E_A3        : constant EBCDIC_CHARACTER := 't';
E_A4        : constant EBCDIC_CHARACTER := 'u';
E_A5        : constant EBCDIC_CHARACTER := 'v';
E_A6        : constant EBCDIC_CHARACTER := 'w';
E_A7        : constant EBCDIC_CHARACTER := 'x';
```

```
E_A8        : constant EBCDIC_CHARACTER = 'y';
E_A9        : constant EBCDIC_CHARACTER = 'z';
E_AD        : constant EBCDIC_CHARACTER = '[';
E_BD        : constant EBCDIC_CHARACTER = ']';
E_C0        : constant EBCDIC_CHARACTER = '{';
E_C1        : constant EBCDIC_CHARACTER := 'A';
E_C2        : constant EBCDIC_CHARACTER := 'B';
E_C3        : constant EBCDIC_CHARACTER := 'C';
E_C4        : constant EBCDIC_CHARACTER := 'D';
E_C5        : constant EBCDIC_CHARACTER := 'E';
E_C6        : constant EBCDIC_CHARACTER := 'F';
E_C7        : constant EBCDIC_CHARACTER := 'G';
E_C8        : constant EBCDIC_CHARACTER := 'H';
E_C9        : constant EBCDIC_CHARACTER := 'I';
E_D0        : constant EBCDIC_CHARACTER := '}';
E_D1        : constant EBCDIC_CHARACTER := 'J';
E_D2        : constant EBCDIC_CHARACTER := 'K';
E_D3        : constant EBCDIC_CHARACTER := 'L';
E_D4        : constant EBCDIC_CHARACTER := 'M';
E_D5        : constant EBCDIC_CHARACTER := 'N';
E_D6        : constant EBCDIC_CHARACTER := 'O';
E_D7        : constant EBCDIC_CHARACTER := 'P';
E_D8        : constant EBCDIC_CHARACTER := 'Q';
E_D9        : constant EBCDIC_CHARACTER := 'R';
E_E0        : constant EBCDIC_CHARACTER := '\';
E_E2        : constant EBCDIC_CHARACTER := 'S';
E_E3        : constant EBCDIC_CHARACTER := 'T';
E_E4        : constant EBCDIC_CHARACTER := 'U';
E_E5        : constant EBCDIC_CHARACTER := 'V';
E_E6        : constant EBCDIC_CHARACTER := 'W';
E_E7        : constant EBCDIC_CHARACTER := 'X';
E_E8        : constant EBCDIC_CHARACTER := 'Y';
E_E9        : constant EBCDIC_CHARACTER := 'Z';
E_F0        : constant EBCDIC_CHARACTER := '0';
E_F1        : constant EBCDIC_CHARACTER := '1';
E_F2        : constant EBCDIC_CHARACTER := '2';
E_F3        : constant EBCDIC_CHARACTER := '3';
E_F4        : constant EBCDIC_CHARACTER := '4';
E_F5        : constant EBCDIC_CHARACTER := '5';
E_F6        : constant EBCDIC_CHARACTER := '6';
E_F7        : constant EBCDIC_CHARACTER := '7';
E_F8        : constant EBCDIC_CHARACTER := '8';
E_F9        : constant EBCDIC_CHARACTER := '9';

type EBCDIC_STRING is array (POSITIVE range <>) of EBCDIC_CHARACTER;

function ASCII_TO_EBCDIC (S : STRING) return EBCDIC_STRING;

-- CONSTRAINT_ERROR is raised if E_STRING'LENGTH /= A_STRING'LENGTH;
procedure ASCII_TO_EBCDIC (A_STRING : in STRING;
                           E_STRING : out EBCDIC_STRING);
```

```
function EBCDIC_TO_ASCII (S : EBCDIC_STRING) return STRING;

-- CONSTRAINT_ERROR is raised if E_STRING'LENGTH /= A_STRING'LENGTH.
procedure EBCDIC_TO_ASCII (E_STRING : in EBCDIC_STRING;
                           A_STRING : out STRING);

end EBCDIC;
```

The procedures ASCII_TO_EBCDIC and EBCDIC_TO_ASCII are much more
efficient than the corresponding functions, as they do not make use of the program
heap. Note that if the **in** and **out** string parameters are of different lengths (i.e.
A_STRING'LENGTH /= E_STRING'LENGTH), the procedures will raise the
exception CONSTRAINT_ERROR.

8.4 Package OS_ENV

The implementation-defined package OS_ENV enables an Ada program to
communicate with the environment in which it is executed.

The specification of this package is as follows:

```
package OS_ENV is

        subtype EXIT_STATUS is INTEGER;

        function ARG_LINE return STRING;

        procedure ARG_LINE (LINE : out STRING;
                            LAST : out NATURAL);

        function ARG_START return NATURAL;

        procedure SET_EXIT_STATUS (STATUS : in EXIT_STATUS);

        procedure ABORT_PROGRAM (STATUS : in EXIT_STATUS);

    end OS_ENV;
```

The exit status of the program (returned in register 15 on exit) can be set by a call
of SET_EXIT_STATUS. Subsequent calls of SET_EXIT_STATUS will overwrite
the exit status which is by default 0. If SET_EXIT_STATUS is not called, a
positive exit code may be set by the Ada Run-Time Executive if an unhandled
exception is propagated out of the main subprogram, or if a deadlock situation is
detected.

The following exit codes relate to unhandled exceptions:

Exception	Code	Cause of exception
NUMERIC_ERROR:		
	1	divide by zero

	2	numeric overflow
CONSTRAINT_ERROR:		
	3	discriminant error
	4	lower bound index error
	5	upper bound index error
	6	length error
	7	lower bound range error
	8	upper bound range error
	9	*null access value*
STORAGE_ERROR:		
	10	frame overflow
		(overflow on subprogram entry)
	11	stack overflow
		(overflow otherwise)
	12	heap overflow
PROGRAM_ERROR:		
	13	access before elaboration
	14	function left without return
SPURIOUS_ERROR:		
	15-20	<an erroneous program>
NUMERIC_ERROR	21	(other than for the above reasons)
CONSTRAINT_ERROR	22	(other than for the above reasons)
	23-24	<unused>
	25	static exception
		(any exception raised as the
		result of a raise statement)

Code 100 is used if a deadlocking situation is detected and the program is aborted as a result.

Codes 1000-1999 are used to indicate other anomalous conditions in the initialisation of the program, messages concerning which are displayed on the terminal.

9   Characteristics of Numeric Types

9.1   Integer Types

The ranges of values for integer types declared in package STANDARD are as follows:

| SHORT_INTEGER | -32768 .. 32767 | -- $2^{**}15 - 1$ |
| INTEGER | -2147483648 .. 2147483647 | -- $2^{**}31 - 1$ |

For the packages DIRECT_IO and TEXT_IO, the ranges of values for types COUNT and POSITIVE_COUNT are as follows:

COUNT	0 .. 2147483647	-- 2**31 - 1
POSITIVE_COUNT	1 .. 2147483647	-- 2**31 - 1

For the package TEXT_IO, the range of values for the type FIELD is as follows:

FIELD	0 .. 255	-- 2**8 - 1

9.2   Floating Point Type Attributes

SHORT_FLOAT

DIGITS	6
MANTISSA	21
EMAX	84
EPSILON	2.0 ** -20
SMALL	2.0 ** -85
LARGE	2.0 ** 84 * (1.0 - 2.0 ** -21)
SAFE_EMAX	252
SAFE_SMALL	2.0 ** -253
SAFE_LARGE	2.0 ** 127 * (1.0 - 2.0 ** -21)
FIRST	-2.0 ** 252 * (1.0 - 2.0 ** -24)
LAST	2.0 ** 252 * (1.0 - 2.0 ** -24)
MACHINE_RADIX	16
MACHINE_MANTISSA	6
MACHINE_EMAX	63
MACHINE_EMIN	-64
MACHINE_ROUNDS	FALSE
MACHINE_OVERFLOWS	TRUE
SIZE	32

FLOAT

DIGITS	15
MANTISSA	51
EMAX	204
EPSILON	2.0 ** -50
SMALL	2.0 ** -205
LARGE	2.0 ** 204 * (1.0 - 2.0 ** -51)
SAFE_EMAX	252
SAFE_SMALL	2.0 ** -253
SAFE_LARGE	2.0 ** 252 * (1.0 - 2.0 ** 51)
FIRST	-2.0 ** 252 * (1.0 - 2.0 ** -56)
LAST	2.0 ** 252 * (1.0 - 2.0 ** -56)
MACHINE_RADIX	16
MACHINE_MANTISSA	14
MACHINE_EMAX	63
MACHINE_EMIN	-64
MACHINE_ROUNDS	FALSE

```
MACHINE_OVERFLOWS              TRUE
SIZE                           64


LONG_FLOAT

DIGITS                         18
MANTISSA                       61
EMAX                           244
EPSILON                        2.0 ** -60
SMALL                          2.0 ** -245
LARGE                          2.0 ** 244 * (1.0 - 2.0 ** -61)
SAFE_EMAX                      252
SAFE_SMALL                     2.0 ** -253
SAFE_LARGE                     2.0 ** 252 * (1.0 - 2.0 ** -61)
FIRST                          -2.0 ** 252 * (1.0 - 2.0 ** -112)
LAST                           2.0 ** 252 * (1.0 - 2.0 ** -112)
MACHINE_RADIX                  16
MACHINE_MANTISSA               28
MACHINE_EMAX                   63
MACHINE_EMIN                   -64
MACHINE_ROUNDS                 FALSE
MACHINE_OVERFLOWS              TRUE
SIZE                           128
```

9.3 Attributes of Type DURATION

```
DURATION'DELTA                 2.0 ** -14
DURATION'SMALL                 2.0 ** -14
DURATION'LARGE                 131072.0
DURATION'FIRST                 -86400.0
DURATION'LAST                  86400.0
```

10 Other Implementation-Dependent Characteristics

10.1 Characteristics of the Heap

All objects created by allocators go into the heap. Also, portions of the Ada Run-Time Executive's representation of task objects, including the task stacks, are allocated in the heap.

All objects whose visibility is linked to a subprogram or block have their storage reclaimed at exit.

Use of UNCHECKED_DEALLOCATION on a task object may lead to unpredictable results.

10.2 Characteristics of Tasks

The default task stack size is 16 Kbytes, but by using the Binder option TASK the size for all task stacks in a program may be set to any size from 4 Kbytes to 16 Mbytes.

Timeslicing is implemented for task scheduling. The default time slice is 1000 milliseconds, but by using the Binder option SLICE the time slice may be set to any period of 10 milliseconds or more. It is also possible to use this option to specify no timeslicing, i.e. tasks are scheduled only at explicit synchronisation points. Timeslicing is started only upon activation of the first task in the program, so the SLICE option has no effect for sequential programs.

Normal priority rules are followed for preemption, where PRIORITY values run in the range 1 .. 10. All tasks with "undefined" priority (no pragma PRIORITY) are considered to have a priority of 0.

The minimum timeable delay is 10 milliseconds.

The maximum number of active tasks is limited only by memory usage. Tasks release their storage allocation as soon as they have terminated.

The acceptor of a rendezvous executes the accept body code in its own stack. A rendezvous with an empty accept body (e.g. for synchronisation) does not cause a context switch.

The main program waits for completion of all tasks dependent on library packages before terminating. Such tasks may select a terminate alternative only after completion of the main program.

Abnormal completion of an aborted task takes place immediately, except when the abnormal task is the caller of an entry that is engaged in a rendezvous. Any such task becomes abnormally completed as soon as the rendezvous is completed.

If a global deadlock situation arises because every task (including the main program) is waiting for another task, the program is aborted and the state of all tasks is displayed.

10.3 Definition of a Main Program

A main program must be a non-generic, parameterless, library procedure.

10.4 Ordering of Compilation Units

The Alsys IBM 370 Ada Compiler imposes no additional ordering constraints on compilations beyond those required by the language. However, if a generic unit is instantiated during a compilation, its body must be compiled prior to the completion of that compilation [10.3].

11 Limitations

11.1 Compiler Limitations

- The maximum identifier length is 255 characters.

- The maximum line length is 255 characters.

- The maximum number of unique identifiers per compilation unit is 1500.

- The maximum number of compilation units in a library is 1023.

- The maximum number of subunits per compilation unit is 100.

- The maximum size of the generated code for a single program unit (subprogram or task body) is 128 Kbytes.

- There is no limit (apart from machine addressing range) on the size of the generated code for a single compilation unit.

- There is no limit (apart from machine addressing range) on the size of a single array or record object.

- The maximum size of a single stack frame is 64 Kbytes including the data for inner package subunits which is "unnested" to the parent frame.

- The maximum amount of data in the global data area of a single compilation unit is 64 Kbytes, including compiler-generated data.

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

NAME AND MEANING   VALUE

$BIG_ID1   A....A1
  Identifier the size of the |----|
  maximum input line length   254 characters
  with varying last character.

$BIG_ID2   A....A2
  Identifier the size of the |----|
  maximum input line length   254 characters
  with varying last character.

$BIG_ID3   A....A3A....A
  Identifier the size of the |----| |----|
  maximum input line length   127   127 characters
  with varying middle character.

$BIG_ID4   A....A4A....A
  Identifier the size of the |----| |----|
  maximum input line length   127   127 characters
  with varying middle character.

$BIG_INT_LIT   0....0298
  An integer literal of value 298 |----|
  with enough leading zeroes so   252 characters
  that is is the size of the
  maximum line length.

$BIG_REAL_LIT   0....069.0E1
  A real literal that can be |----|
  either of floating- or fixed-   249 characters
  point type, has value of 690.0,
  and has enough leading zeroes to
  be the size of the maximum line
  length.

NAME AND MEANING	VALUE

$BLANKS
  A sequence of blanks twenty
  characters fewer than the size
  of the maximum line length.

235 blanks

$COUNT_LAST
  A universal integer literal
  whose value is TEXT_IO.COUNT'LAST.

2_147_483_647

$EXTENDED_ASCII_CHARS
  A string literal containing all
  the ASCII characters with
  printable graphics that are not
  in the basic 55 Ada character
  set.

"abcdefhgijklmnopqrstuvwxyz
!$%?@[\]^'{}~"

$FIELD_LAST
  A universal integer literal
  whose value is TEXT_IO.FIELD'LAST

255

$FILE_NAME_WITH_BAD_CHARS
  An illegal external file name
  that either contains invalid
  characters or is too long if no
  invalid characters exist.

T??????? LISTING A1

$FILE_NAME_WITH_WILD_CARD_CHAR
  An external file name that
  either contains a wild card
  character or is too long if no
  wild card characters exists.

TOOLONGNAME LISTING A1

$GREATER_THAN_DURATION
  A universal real value that lies
  between DURATION'BASE'LAST and
  DURATION'LAST if any, otherwise
  any value in in the range of
  DURATION.

100_000.0

$GREATER_THAN_DURATION_BASE_LAST
  The universal real value that is
  greater than DURATION'BASE'LAST,
  if such a value exists.

10_000_000.0

$ILLEGAL_EXTERNAL_FILE_NAME1
  An illegal external file name.

T???????

NAME AND MEANING	VALUE
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	TOOLONGNAME LISTING A1
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2_147_483_648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2_147_483_647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10_000_000.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	18
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	255
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	NO_SUCH_TYPE

C-3

NAME AND MEANING	VALUE
$NEG_BASED_INT A based integer literal whose highest order non-zero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	8#20000000000#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non_ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

. C32114A: An unterminated string literal occurs at line 62.

. B33203C: The reserved word "IS" is misspelled at line 45.

. C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.

. C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.

. B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.

. C41404A: The values of 'LAST and 'LENGTH are incorrect in the _if_ statements from line 74 to the end of the test.

. B45116A: ARRPRIBL 1 and ARRPRIBL 2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.

. C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.

. B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and _end_ _case_; is missing from line 42.

. B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

D-1

. B74101B: The begin at line 9 causes a declarative part to be
                  treated as a sequence of statements.

. C87B50A: The call of "/=" at line 31 requires a use clause
                  for package A.

. C92005A: The "/=" for type PACK.BIG_INT at line 40 is not
                  visible without a use clause for the package PACK.

. C940ACA: The assumption that allocated task TT1 will run
                  prior to the main program, and thus assign SPYNUMB
                  the value checked for by the main program, is
                  erroneous.

. CA3005A..D: No valid elaboration order exists for these tests.
  (4 tests)

. BC3204C: The body of BC3204C0 is missing.

.

END

DATE

FILMED

MARCH

1988

DTIC